

video 1: miscellaneous topic: Delegating constructors

Delegating Constructors

2 Review: Constructors

```
class Triangle {
private:
    double a;
    double b;
    double c;
public:
    Triangle(double a_in, double b_in, double c_in) : a(a_in), b(b_in), c(c_in) {
        // Nothing to do in function body.
        // This isn't always the case.
    }
    Triangle() : a(1), b(1), c(1) { }
}
```

Parameters receive arguments provided when making a Triangle object.

A special syntax for initializing members in a constructor.

A default constructor takes no arguments.

Same "name" as the class.

Triangle t(3,4,5);

3 Delegating Constructors

```
class Triangle {
...
    Triangle(double a_in, double b_in, double c_in) : a(a_in), b(b_in), c(c_in) {
        // Be assertive!
        assert(a > 0 && b > 0 && c > 0);
        assert(a + b > c && a + c > b && b + c > a);
    }
    Triangle(double side_in) : a(side_in), b(side_in), c(side_in) {
        // Be assertive!
        assert(a > 0 && b > 0 && c > 0);
        assert(a + b + c == 2 * side_in);
    }
    Triangle(double side_in) : Triangle(side_in, side_in, side_in) {
        Better
    }
}
```

Check that the inputs will make a valid Triangle.

Code duplication!

Is this a good approach?

NO.

MORE VIDEOS

Delegating Constructors

```
class Triangle {
...
    Triangle(double a_in, double b_in, double c_in) : a(a_in), b(b_in), c(c_in) {
        // Be assertive!
        assert(a > 0 && b > 0 && c > 0);
        assert(a + b > c && a + c > b && b + c > a);
    }
    Triangle(double side_in) : a(side_in), b(side_in), c(side_in) {
        // Be assertive!
        assert(a > 0 && b > 0 && c > 0);
        assert(a + b + c == 2 * side_in);
    }
    Triangle(double side_in) : Triangle(side_in, side_in, side_in) {
        Compiler basically sees:
        Triangle temp(side_in, side_in, side_in);
    }
}
```

DO

IT'S A TRAP

DON'T

Delegation has to be done in an initializer list.

Compiler basically sees: Triangle temp(side_in, side_in, side_in);

MORE VIDEOS

Video 2: introduction to inheritance

Inheritance Basics

8 Lots of code duplication!

C++ Style Bird ADTs

```
class Chicken {
private:
    int age;
    string name;
    int roadsCrossed;
public:
    Chicken(const string &name_in) : age(0), name(name_in), roadsCrossed(0) {
        cout << "Chicken ctor" << endl;
    }
    string getName() const { return name; }
    int getAge() const { return age; }
    void crossRoad() {
        ++roadsCrossed;
    }
    void talk() const {
        cout << "bawk" << endl;
    }
}
```

```
class Duck {
private:
    int age;
    string name;
    int numDucklings;
public:
    Duck(const string &name_in) : age(0), name(name_in), numDucklings(0) {
        cout << "Duck ctor" << endl;
    }
    string getName() const { return name; }
    int getAge() const { return age; }
    void babyDucklings() {
        numDucklings += 7;
    }
    void talk() const {
        cout << "quack" << endl;
    }
}
```

Is this a good approach?

NO.

We've got lots of code because of the inheritance.

MORE VIDEOS

9 Inheritance

- Consider ADTs that represent birds...
- Intuitively, Chicken and Duck ADTs are both specific kinds of Birds.

```
Bird
+--> Chicken
+--> Duck
```

We will use Bird as a base type, with Chicken and Duck as derived types.

This is called inheritance.

MORE VIDEOS

10 Inheritance with Birds

Birds and Inheritance

Base Class

```
class Bird {
private:
    int age;
    string name;
public:
    Bird(const string &name_in) : age(0), name(name_in) {
        cout << "Bird ctor" << endl;
    }
    string getName() const { return name; }
    int getAge() const { return age; }
    void haveBirthday() { ++age; }
    void talk() const {
        cout << "tweet" << endl;
    }
}
```

Derived Class

```
class Chicken : public Bird {
private:
    int roadsCrossed;
public:
    Chicken(const string &name_in) : Bird(name_in), roadsCrossed(0) {
        cout << "Chicken ctor" << endl;
    }
    void crossRoad() { ++roadsCrossed; }
    void talk() const {
        cout << "bawk" << endl;
    }
}
```

Bird ctor Chicken ctor

We want to look up a member named talk.

MORE VIDEOS

11 Birds and Ducks

Base Class

```
class Bird {
private:
    int age;
    string name;
public:
    Bird(const string &name_in) : age(0), name(name_in) {
        cout << "Bird ctor" << endl;
    }
    string getName() const { return name; }
    int getAge() const { return age; }
    void haveBirthday() { ++age; }
    void talk() const {
        cout << "tweet" << endl;
    }
}
```

Derived Class

```
class Duck : public Bird {
private:
    int numDucklings;
public:
    Duck(const string &name_in) : Bird(name_in), numDucklings(0) {
        cout << "Duck ctor" << endl;
    }
    void babyDucklings() {
        numDucklings += 7;
    }
    void talk() const {
        cout << "quack" << endl;
    }
}
```

with hierarchy mention

MORE VIDEOS

12 Calling a Base-Class Ctor

- A constructor in a derived class always calls a constructor for a base class.
- If no explicit call is made in the member initializer list, the default constructor of the base is implicitly called.

```
class Chicken : public Bird {
    Chicken(const string &name_in) : Bird(name_in), roadsCrossed(0) {}
    ...
    OK: explicit call to base ctor
};

class Chicken : public Bird {
    Chicken() : roadsCrossed(0) {}
    ...
    Error: base class has no default ctor
};

class Square : public Rectangle {
    Square() {}
    ...
    OK: implicit call to default base ctor
}
```

MORE VIDEOS

13 Varieties of Inheritance

- C++ offers three different varieties of inheritance:
 - public
 - protected
 - private
- We only cover public inheritance. Don't worry about the other ones.

```
class Duck : public Bird {
    ...
    ...
};

Specify which kind of inheritance you want like this. If left blank, defaults to private.
```

MORE VIDEOS

19 Inheritance and Memory

In memory, a derived class object has a base class part.

You don't have to do anything special to access it.

Member access with . or -> will find base members too.

```
int main() {
    Chicken c("Myrtle");
    Duck d("Scrooge");
    Bird b("Big Bird");
    c.age; // automatically refers to the age member in the Bird part of the Chicken c.
}
```

Technically doesn't work here since age is private, though.

REVIDEOS

2/18/2021

20 Compound Object Lifetimes

- Constructors are called whenever a class object is created for the first time.
- destructors are called whenever a class object's lifetime ends (depends on storage duration).
 - for local variables, when they go out of scope
- We will talk about destructors in more detail later.

```
Triangle()
: a(1), b(1), c(1) {
    cout << "Triangle ctor" << endl;
}

~Triangle() {
    cout << "Triangle dtor" << endl;
}
```

like when scope ends

MORE VIDEOS

2/18/2021

21 Ctors and Dtors in Derived Types

- Destructors are the analog of constructors, called when an object is destroyed.
- When creating or destroying an object of a class with a base type, a constructor or destructor is used for each level of the hierarchy.
- For constructors, we get top-down behavior.

```
int main() {
    Duck d("Scrooge"); // Animal ctor, Bird ctor, Duck ctor
    Bird b("Big Bird"); // Animal ctor, Bird ctor
    ...
}
```

is it pure okay zone

- For destructors, we get bottom-up behavior.

```
...
~d ~Bird dtor, Animal dtor
~b ~Duck dtor, Bird dtor, Animal dtor
~d ~Duck dtor, Bird dtor, Animal dtor
```

MORE VIDEOS

2/18/2021

22 Member Name Lookup

- When we use . or -> for member access, how does the compiler look up the member's name?
- Start in the first class scope.
- If not found, try base class scope as well.
- Stop whenever a matching name is found.

```
class Bird {
private:
    int age;
    string name;
public:
    void talk() const {
        cout << "Tweet" << endl;
    }
    void getName() const;
    int getAge() const;
};

class Chicken : public Bird {
private:
    int roadsCrossed;
public:
    Chicken(const string &name_in) : Bird(name_in), roadsCrossed(0) {
        cout << "Chicken ctor" << endl;
    }
    void crossRoad() { ++roadsCrossed; }
    void talk() const {
        cout << "Bawk" << endl;
    }
}
```

We want to look up a member named talk.

MORE VIDEOS

2/18/2021

23 Name Hiding

- When we use . or -> for member access, how does the compiler look up the member's name?
- Start in the first class scope.
- If not found, try base class scope as well.
- Stop whenever a matching name is found.

```
class Bird {
private:
    void talk() const {
        cout << "Tweet" << endl;
    }
    string getName() const;
    int getAge() const;
};

class Chicken : public Bird {
private:
    void talk() const {
        cout << "Bawk" << endl;
    }
    ...
}
```

Hidden

We want to look up a member named talk.

MORE VIDEOS

2/18/2021

24 Name Hiding

- Name hiding can have tricky consequences.
- ONLY the name, not the signature, is considered for name lookup.

```
class Bird {
private:
    void talk() const {
        cout << "Tweet" << endl;
    }
    string getName() const;
    int getAge() const;
};

class Chicken : public Bird {
private:
    void talk(int volumeLevel) const {
        if (volumeLevel > 0) {
            cout << "BawWWK" << endl;
        } else {
            cout << "Bawk" << endl;
        }
    }
}
```

Hidden

We want to look up a member named talk.

Compiles error! Parameter types don't match.

MORE VIDEOS

2/18/2021

25 Accessing a Hidden Name

- We can use the scope resolution operator to access a hidden name (i.e. by using a qualified name).

```
class Bird {
private:
    int age;
    string name;
public:
    ...
    void talk() const {
        cout << "Tweet" << endl;
    }
    string getName() const;
    int getAge() const;
};

class Chicken : public Bird {
private:
    int roadsCrossed;
public:
    Chicken(const string &name_in) : Bird(name_in), roadsCrossed(0) {
        cout << "Chicken ctor" << endl;
    }
    void crossRoad() { ++roadsCrossed; }
    void talk() const {
        cout << "Bawk" << endl;
    }
}
```

Problem: age is private in Bird

MORE VIDEOS

2/18/2021

26 The protected Access Level

- We can make a member accessible to derived classes using protected instead of private.

```
class Bird {
protected:
    int age;
    string name;
public:
    ...
    void talk() const {
        cout << "Tweet" << endl;
    }
};

class Chicken : public Bird {
private:
    int roadsCrossed;
public:
    ...
    void talk() const {
        if (age > 1) {
            cout << "Bawk" << endl;
        } else {
            // baby chicks tweet
            Bird::talk();
        }
    }
}
```

Problem: this reveals implementation details of Bird

MORE VIDEOS

2/18/2021

27 Better Solution

- Use a public "getter" function instead.

```
class Bird {
private:
    int age;
    string name;
public:
    ...
    int getAge() const {
        return age;
    }
    void talk() const {
        cout << "Tweet" << endl;
    }
};

class Chicken : public Bird {
private:
    int roadsCrossed;
public:
    ...
    void talk() const {
        if (getAge() >= 1) {
            cout << "Bawk" << endl;
        } else {
            // baby chicks tweet
            Bird::talk();
        }
    }
}
```

age if we want to change

MORE VIDEOS

2/18/2021